



PATENT ABSTRACTS OF JAPAN

(11) Publication number: 2000020354 A

(43) Date of publication of application: 21 . 01 . 00

(51) Int. Cl.

G06F 11/34
G06F 11/28

(21) Application number: 10191918

(22) Date of filing: 07 . 07 . 98

(71) Applicant: HITACHI LTD

(72) Inventor: AKITA SHUNICHI
NOMURA YUJI

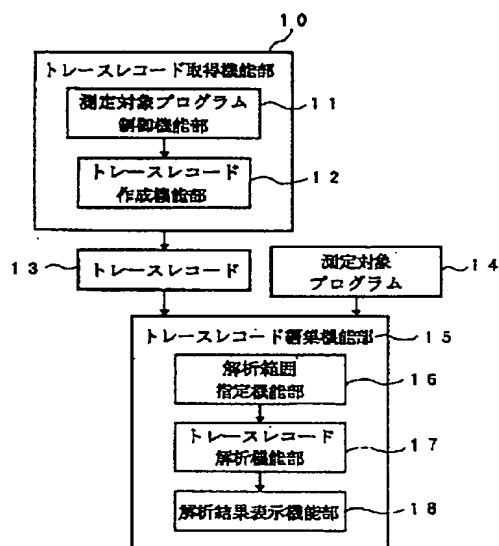
(54) EDITOR FOR NUMBER OF EXECUTION STEPS

COPYRIGHT: (C)2000,JPO

(57) Abstract:

PROBLEM TO BE SOLVED: To designate an analysis range on a function call related diagram by displaying the total of how many times respective functions are called, the number of execution steps and the execution step numbers of all the functions transited from that function on the function call related drawing.

SOLUTION: A measuring object program control function part 11 executes a program, generated interruption for each branching instruction, performs the preparation/output of trace record and prepares a file 13. In the file 13, information such as the kind of event, event generation address and event generation instruction is stored so as to output the number of execution steps, function transition condition, interruption generation and system call issue condition or the like at the time of editing. An editing function part 15 designates the editing range by inputting the trace record in the file 13 and the measuring object program 14 and prepares the function transition diagram and a function call related diagram by performing analytic processing.



This Page Blank (uspto)

Our Ref: OP1135-US

Prior Art Reference:

Japanese Patent Laid-Open Publication No. 2000-20354
Laid-Open Date: January 21, 2000
Title: DEVICE FOR EDITING THE NUMBER OF EXECUTION STEPS
Patent Application No. Hei 10-191918
Filing Date: July 7, 1998
Applicant: ID No. 000005108
KABUSHIKI KAISHA HITACHI SEISAKUSHO
Chiyoda-ku, Tokyo, Japan
Inventors: Shunichi AKITA and Yuuji NOMURA
both c/o Software Development Center of
Kabushiki Kaisha Hitachi Seisakusho
Yokohama-shi, Kanagawa-ken, Japan

(Partial Translation)

[ABSTRACT]

[Object] Capable of specifying a range of analysis on a function call relation diagram, by displaying the number of calls of each function, the number of execution steps and a total number of execution steps of all functions which have been made to transit, on the function call relation diagram.

[Solving Means] An object program measurement control function section 11 performs execution of a program, generation of interruption at each branch instruction, preparation and outputting of a trace record, and preparation of a file 13. The file 13 is stored with information including types of event, event generation addresses and event generation instructions so that the number of execution steps, the state of function transition, generation of interruption and the state of issuing system calls can be outputted at the time of editing. An editing function section 15 has, as its input, the trace record from the

This Page Blank (uspto)

file 13 and the object program 14 to be measured, to specify a range of editing and to perform an analytic process, thereby to prepare a function transition diagram and a function call relation diagram.

- - - - -

[0013]

Fig. 1 is a block diagram showing a whole structure of an editing device for the number of execution steps according to an embodiment of the present invention; Fig. 2 is an explanatory diagram of a structure of a function transition diagram; Fig. 3 is a flowchart explaining an operation of preparing the function transition diagram; Fig. 4 is a diagram explaining a structure of a function call relation diagram; Fig. 5 is a flowchart explaining an operation of preparing the function call relation diagram; Fig. 6 is a flowchart explaining an output operation of the function call relation diagram; Fig. 7 is a diagram explaining a process of specifying a range of analysis; and Fig. 8 is a flowchart explaining an execution operation of the analysis in the specified analysis range. In Fig. 1, numeral 10 is a trace record acquisition function section, 11 is an object program measurement control function section, 12 is a trace record preparing function section, 13 is a trace record file, 14 is an object program to be measured, 15 is a trace record editing function section, 16 is an analysis range specifying function section, 17 is a trace record analyzing function section, and 18 is an analytic result display function section.

[0014]

The editing device for the number of execution steps according to an embodiment of the present invention is generally consisting of, as shown in Fig. 1, the trace record acquisition function section 10 and the trace record editing function section

This Page Blank (uspto)

15. The trace record acquisition function section 10 further comprises the object program measurement control function section 11 and the trace record preparing function 12. And, the trace record editing function section 15 comprises the analysis range specifying function section 16, the trace record analyzing function section 17 and the analytic result display function section 18.

[0015]

The object program measurement control function section 11 of the trace record acquisition function section 10 causes execution of the program, of which the number of execution steps is to be measured, under the control by the function section 11 and causes generation of interruption at each branch instruction. The object program measurement control function section 11 monitors the generation of interruption, and when an interruption is generated, calls the trace record preparing function section 12 and causes it to prepare and output a trace record, thereby to prepare the trace record file 13. This trace record file 13 is stored with information including types of event, event generation addresses and event generation instructions so that the number of execution steps, the state of function transition, generation of interruptions and the state of issuing system calls can be output at the time of editing by the trace record editing function section 15.

[0016]

The trace record editing function section 11 executes the editing process having, as its input, the trace record in the trace record file 13 and the configuration information of the object program 14 to be measured. With this editing process, first, the editing range is specified by the analysis range specifying function section 16. Next, upon performing the analytic process by the trace record analyzing function section 17, the function transition diagram as shown in Fig. 2 and the

This Page Blank (uspto)

function call relation diagram as shown in Fig, 4 are prepared, and further, the diagram prepared are displayed by the trace record display function section 18. The process is executed in this sequence.

[0017]

The function transition diagram shown in Fig. 2 is a table of functions executed in a time series sequence wherein a nest 20 of function, a function name 21, and the number 22 of steps executed by each function, are outputted. In this example of Fig. 2, a function `fanca` performs the process in 85 steps until a function `fancb` is called, the function `fancb` performs the process in 101 steps during a period of from the time when it was called by the function `fanca` until a function `fancc` is called, and the function `fancc` performs the process in 87 steps during a period from the time when it was called by the function `fancb` until it is returned to the function `fancb`. A further description of the function transition will be omitted, but the function transition diagram shows the state of transition of functions related to sequential calls and the number of nests of the return function, and also shows the number of process steps thereof.

[0018]

Next, the operation of preparing the function transition diagram by the trace record analyzing function section 17 will be described by referring to the flowchart shown in Fig. 3.

[0019]

(1) First, read-in the initial trace record, compares the address in the trace record with a corresponding table between the function address and the function name obtained from the object program to be measured, thereby to acquire the function name and to prepare a table 30 for analysis. The table 30 for analysis is a table for temporary storing the information to be used for analysis. As shown in Fig. 3, the table 30 is formed of a record consisting of the function name, the number of

This Page Blank (uspto

display nests, and the number of execution steps. First, initialization is set as: the number of display nests=1, and the number of execution steps=0 (Steps 31, 32).

[0020]

(2) Read-in the next trace record, check whether the trace record has been successfully acquired or not. When the next trace record does not exist, this process is terminated (Steps 33, 34).

[0021]

(3) When it is determined in Step 34 that the trace record has been successfully acquired, the number of execution steps is calculated from the difference in the addresses between the trace records, and the calculated result is added to the number of execution steps in the record of the table 30 for analysis which was prepared in Step 32 (Step 35).

[0022]

(4) From the branch destination address of the trace record acquired in Step 33 and from a branch instruction, it is determined whether there are any function calls. When there is a function call, each information of the table for analysis which has been prepared up-to that point is outputted. Thereafter, the function name of the call destination is acquired from the branch destination address in the trace record and from the corresponding table of the function address and function name, thereupon a new table 30 for analysis is prepared. For the information in the record of the table for analysis, other than the function names, the number of display nests is set to: the previous number of display nests + 1, and the number of execution steps is set to: 0 (Steps 36, 37).

[0023]

(5) When it is determined in Step 36 that there is no function call, then whether it is the function return or not is determined from the branch destination address and the branch

This Page Blank (uspto)

instruction. If it is the function return, each information for the table for analysis which has been prepared up-to that point is outputted. Thereafter, the function name of the return destination is acquired from the branch destination address in the trace record and from the corresponding table of the function address and the function name, whereupon a new table 30 for analysis is prepared. For the information in the record of the table for analysis, other than the function name, the number of display nests is set to: the previous number of display nests - 1, and the number of execution steps is set to: 0 (Steps 38, 39).
[0024]

(6) After finishing the process of Step 39 and when determined in Step 38 that it is not the function return, the processes from Step 33 are repeated.
[0025]

The function call relation diagram is, as shown in Fig. 4, a table wherein a function name 40, the number 41 of calls of each function, the number 43 of the execution steps of each function, and a total number 42 of execution steps of the functions subsequently called, are outputted based on the function transition diagram. In Fig. 4, according to the number of nests of each function, the number 43 of the execution steps of the associated function and also the total number 42 of the execution steps of the functions subsequently called are shifted to the right.

[0026]

The function call relation diagram is prepared according to the flowchart shown in Fig. 5. The process thereof will be described below by referring to Fig. 5.

[0027]

(1) Read-in the initial trace record, compares the address in the trace record with a corresponding table between the function address and the function name obtained from the object

This Page Blank (1)

program to be measured, thereby to acquire the function name, and prepare a starting node. Then, thus prepared starting node is to be the current node which is the node to be processed (Steps 51, 52).

[0028]

Note that the node being used here is designated as a node 50 as shown in Fig. 5. This node is a table for storing information for the purpose of holding the information of each function which appears (name of function, the number of execution steps, etc.), and the node is prepared in a manner of one node per one function and in the form of a bidirectional list which can have a plurality of child nodes. The node 50 has the information including a pointer ***pre** for a previous node, the function name **name**, the number of calls **call**, the number of execution steps (43 in Fig. 4) **step**, a total number of execution steps (42 in Fig. 4) **astep**, a total number of execution steps **wastep** per one call to be used for internal process, and a pointer ***next [n]** for the next node.

[0029]

In the starting node, the information other than the function name is set as follows.

- .Pointer of the parent (previous) node : none
- .The number of function calls : 1
- .The number of execution steps : 0
- .The total number of execution steps : 0
- .The total number of execution steps (per one call) : 0
- .Pointer of the child (next) node : no connection at all

[0030]

(2) Read-in a new trace record, check whether the trace record has been successfully acquired or not. When the trace record does not exist, analysis is terminated and output process which will be described later will be performed, whereupon this process is finished (Steps 53, 54, 5D).

This Page Blank (usr

[0031]

(3) When the trace record exists in Step 54, the number of execution steps is calculated from the difference in the addresses of the acquired trace records, and the calculated result is added to the number of execution steps of the current node and to the total number of execution steps (per one call) (Step 55).

[0032]

(4) From the branch destination address and the branch instruction, it is determined whether there are any function calls. When there is a function call, the function name of the call destination is acquired from the branch destination address in the trace record and from the corresponding table of the function address and function name, thereupon a search is made having the call destination function as a key to determine whether the call destination function node is linked with the current node (Steps 56, 57).

[0033]

(5) It is determined whether the child node has been found or not by the search performed in Step 57. If the child node has been found, the information in the call destination node is changed as follows, by using the found node as the call destination node.

- . The number of function calls : +1
- . The total number of execution steps (per one call) : 0

After making this change of information, the current node is changed to the call destination node, and the processes from the Step 53 are repeatedly executed (Steps 58, 5A).

[0034]

(6) When the child node is not found in Step 58, a new node is prepared. In the new node, the information, other than the function name, is set as follows.

This Page Blank (us

.Pointer of the parent (previous) node : current node
.The number of function calls : 1
.The number of execution steps : 0
.The total number of execution steps : 0
.The total number of execution steps (per one call) : 0
.Pointer of the child (next) node : no connection at all

After setting the information as such, the current node is changed to a new node, and the processes from Step 53 are executed repeatedly (Step 59).

[0035]

(7) When it is determined in Step 36 that there is no function call, then whether it is the function return or not is determined from the branch destination address and the branch instruction. If it is not the function return, the processes from Step 53 are executed repeatedly (Step 5B).

[0036]

(8) When determined in the Step 5B that there is the function return, the parent node is used as the return destination node. Then, only the information related to the changes in the current node is changed. Here, the total number of execution steps (per one call) is added to the total number of execution steps of the current node. Further, only the information related to the changes in the return destination node is changed. Here, the total number of execution steps (per one call) of the current node is added to the total number of execution steps (per one call) of the return destination node. After making this information change, the current node is changed to the return destination node, and the processes from the Step 53 are executed repeatedly (Step 5C).

[0037]

While the above-described processes have been continued, and when it occurs that in Step 54 the trace record no longer exists, the output process of the function call relation diagram is

This Page Blank (uspic

performed by the Step 5D, thereby the function call relation diagram as shown in Fig. 4 is outputted. The output process by Step D will be described in detail by referring to Fig. 6.

[0038]

(1) Set the current node as the starting node, and output the information of the current node (starting node) (Steps 61, 62).

[0039]

(2) The current node is checked to determined whether there are any child nodes having the data not outputted yet (Step 63).

[0040]

(3) If determined in Step 63 that a child node having the data not outputted exists, the current node is moved to that child node, the information of the moved current node is outputted, and the processes from Step 63 will be continued.

[0041]

(4) If determined in Step 63 that no child node having the data not outputted yet exists, check and determine whether a parent node exists, and if no parent node exists, the output process is finished (Step 65).

[0042]

(5) If determined in Step 65 that a parent node exists, the current node is moved to the parent node, and the processes from Step 63 will be continued (Step 66).

[0043]

In addition to the above-described output process, the output process is executed under the following output rules. In the case where a plurality of child nodes are connected to the same node, the order of outputting the child nodes is from the one having the larger total number of execution steps. The outputting is performed to have a stepped display of the number of steps as shown in Fig. 4, thereby the function call relation can be readily understood when the diagram is viewed.

This Page Blank (uspto)

(19) 日本国特許庁 (JP)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開 2000-20354

(P 2000-20354 A)

(43) 公開日 平成12年1月21日 (2000. 1. 21)

(51) Int. Cl. 7	識別記号	F I	テーマコード (参考)
G 0 6 F 11/34		G 0 6 F 11/34	N 5B042
11/28	3 1 0	11/28	3 1 0 E

審査請求 未請求 請求項の数 2

O L

(全 9 頁)

(21) 出願番号 特願平10-191918

(22) 出願日 平成10年7月7日 (1998. 7. 7)

(71) 出願人 000005108

株式会社日立製作所

東京都千代田区神田駿河台四丁目6番地

(72) 発明者 秋田 俊一

神奈川県横浜市戸塚区戸塚町5030番地 株式会社日立製作所ソフトウェア開発本部内

(72) 発明者 野村 祐治

神奈川県横浜市戸塚区戸塚町5030番地 株式会社日立製作所ソフトウェア開発本部内

(74) 代理人 100078134

弁理士 武 顕次郎

F ターム (参考) 5B042 EA03 FA09 FB08 FB10

FC05 FD03 FD22 FD23 FD24

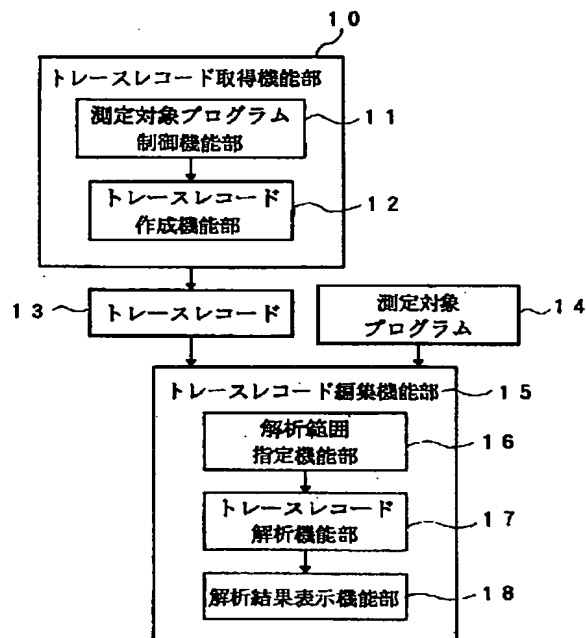
(54) 【発明の名称】 実行ステップ数の編集装置

(57) 【要約】 (修正有)

【課題】 関数呼出し関連図上に各関数の呼び出し回数、実行ステップ数及びその関数から遷移するすべての関数の実行ステップ数の合計を表示し、関数呼出し関連図上で解析範囲を指定することを可能にする。

【解決手段】 測定対象プログラム制御機能部 11 は、プログラムを実行し、分岐命令毎に割り込みを発生させ、トレースレコードの作成・出力を行わせ、ファイル 13 を作成する。ファイル 13 には、編集の際に、実行ステップ数、関数遷移状況、割り込みの発生、システムコールの発行状況等を出力ができるように、イベントの種類、イベント発生アドレス、イベント発生命令等の情報が格納される。編集機能部 15 は、ファイル 13 内のトレースレコードと測定対象プログラム 14 とを入力として、編集範囲の指定を行い、解析処理を行うことにより、関数遷移図、関数呼出し関連図を作成する。

図 1



【特許請求の範囲】

【請求項 1】 ソフトウェアプログラムの実行ステップ数の編集装置において、測定対象プログラムの実行情報を分岐命令の実行及び割り込みの発生を契機にトレースレコードとして時系列順に取得する手段と、測定対象プログラムの構成情報とトレースレコードとから、測定対象プログラム内の特定の関数を基準として、前記関数から遷移する関数の遷移状況と、前記関数から遷移する各関数の呼出し回数と、実行ステップ数と、前記関数から遷移する全ての関数の実行ステップ数の合計との複数の情報を取得する手段と、これらの情報を測定対象プログラムの関数呼出し関連図上に出力する手段とを備えることを特徴とする実行ステップ数の編集装置。

【請求項 2】 前記測定対象プログラムの編集範囲を指定して、前記複数の情報が出力された関数呼出し関連図を作成する手段を備えることを特徴とする請求項 1 記載の実行ステップ数の編集装置。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】 本発明は、プログラムの実行ステップ数の採取と、採取結果の編集とを行う実行ステップ数の編集装置に係り、特に、ソフトウェアプログラムの性能をチェックし、プログラムの性能の向上に役立てるために使用して好適な実行ステップ数の編集装置に関する。

【0002】

【従来の技術】 ソフトウェアプログラムの性能向上のためのプログラムの実行ステップ数の採取は、従来から行われていた。実行ステップ数の取得に関する従来技術として、命令の実行毎に割り込みを発生させて、実行命令数をカウントする方法、分岐命令毎の分岐アドレスから実行命令数を算出する方法等が知られている。

【0003】 また、取得したプログラムの実行ステップ数の編集に関する従来技術として、プログラム全体でのステップ数の表示を行う方法、関数単位でのステップ数を時系列に表示する方法等が知られている。そして、これらの方法は、プログラムの中のどの処理がプログラムの性能のネックとなっているかが、すぐには把握できないものである。

【0004】 なお、プログラムの実行ステップ数の取得方法に関する従来技術として、例えば、特開平 9-34754 号公報等に記載された技術が知られている。

【0005】

【発明が解決しようとする課題】 採取した実行ステップ数の編集を行う前述した従来技術は、時系列順に関数単位で実行ステップ数を関数遷移図上に表示するものであり、プログラム性能のネックとなっている部分を発見するために、時系列順に表示されている表を、手作業によって編集・解析を行わなければならないものである。この手作業による解析作業の 1 つとして、「プログラム内

のどの処理でどれだけの実行ステップ数がかかっているかについて調査するために、処理の起点となる関数が呼出している関数の全ての実行ステップ数の合計を算出する。」という作業があげられるが、この作業を全て手作業で行うためには、多大な時間がかかっていた。また、一般に、プログラムは、同じ処理でも実行のタイミング等により、実行ステップ数に変化することがあるが、従来技術は、そのような違いを把握するためにも、手作業による編集・解析を行い、比較を行っていたため、多大な時間を要している。

【0006】 前述したように従来技術による実行ステップ数の編集は、その殆どを手作業により行わなければならないため、作業者に多大な時間と作業負担をかけるという問題点を有している。

【0007】 本発明の目的は、前記従来技術の問題点を解決し、実行ステップ数の取得結果を関数の呼出し関連図が理解し易い形で出力することを可能にして、性能解析作業を効率良く行うことを可能にし、また、解析範囲の指定を行うことを可能にして、実行タイミングの違いなどによる実行ステップ数の比較を行い易くすることができる実行ステップ数の編集装置を提供することにある。

【0008】

【課題を解決するための手段】 本発明によれば前記目的は、ソフトウェアプログラムの実行ステップ数の編集装置において、測定対象プログラムの実行情報を分岐命令の実行及び割り込みの発生を契機にトレースレコードとして時系列順に取得する手段と、測定対象プログラムの構成情報とトレースレコードとから、測定対象プログラム内の特定の関数を基準として、前記関数から遷移する関数の遷移状況と、前記関数から遷移する各関数の呼出し回数と、実行ステップ数と、前記関数から遷移する全ての関数の実行ステップ数の合計との複数の情報を取得する手段と、これらの情報を測定対象プログラムの関数呼出し関連図上に出力する手段とを備えることにより達成される。

【0009】 また、前記目的は、前記測定対象プログラムの編集範囲を指定して、前記複数の情報が出力された関数呼出し関連図を作成する手段を備えることにより達成される。

【0010】 前述したような構成を有する本発明は、出力する関数呼出し関連図が、トレースレコードを採取した範囲内での関数の呼出し関連図、関数の呼出し回数、関数の実行ステップ数、自関数から呼出している関数群の実行ステップ数の合計の情報を持っているため、手作業により時系列順の表の編集・解析を行う必要がなくなり、関数の呼出し関連等の情報を把握し易くなり、作業者に、効率良くプログラムの性能解析作業を行わせることができる。

【0011】 また、編集範囲の指定により、特定の処理に着目した関数呼出し関連図を出力することが可能とな

り、同一処理の実行タイミングの違い等による実行ステップ数の違いと言った情報も簡単に把握することが可能となる。

【0012】

【発明の実施の形態】以下、本発明による実行ステップ数の編集装置の一実施形態を図面により詳細に説明する。

【0013】図1は本発明の一実施形態による実行ステップ数の編集装置の全体の構成を示すブロック図、図2は関数遷移図の構成を説明する図、図3は関数遷移図の作成動作を説明するフローチャート、図4は関数呼出し関連図の構成を説明する図、図5は関数呼出し関連図の作成動作を説明するフローチャート、図6は関数呼出し関連図の出力動作を説明するフローチャート、図7は解析範囲の指定方法を説明する図、図8は解析範囲を指定した解析の実行動作を説明するフローチャートである。図1において、10はトレースレコード取得機能部、11は測定対象プログラム制御機能部、12はトレースレコード作成機能部、13はトレースレコードファイル、14は測定対象プログラム、15はトレースレコード編集機能部、16は解析範囲指定機能部、17はトレースレコード解析機能部、18は解析結果表示機能部である。

【0014】本発明の一実施形態による実行ステップ数の編集装置は、図1に示すように、大きく分けてトレースレコード取得機能部10とトレースレコード編集機能部15とにより構成されている。トレースレコード取得機能部10は、更に測定対象プログラム制御機能部11とトレースレコード作成機能部12とにより構成されている。また、トレースレコード編集機能部15は、解析範囲指定機能部16と、トレースレコード解析機能部17と、解析結果表示機能部18とにより構成されている。

【0015】トレースレコード取得機能部10の測定対象プログラム制御機能部11は、実行ステップ数の測定対象となるプログラムを、この機能部11の制御下で実行させ、分岐命令毎に割り込みを発生させる。測定対象プログラム制御機能部11は、割り込みの発生を監視し、割り込みが発生すると、トレースレコード作成機能部12を呼び出し、トレースレコード作成機能部12にトレースレコードの作成・出力を行わせ、トレースレコードファイル13を作成する。このトレースレコードファイル13には、トレースレコード編集機能部15で編集を行う際に、実行ステップ数、関数遷移状況、割り込みの発生、システムコールの発行状況等を出力することができるよう、イベントの種類、イベント発生アドレス、イベント発生命令等の情報が格納される。

【0016】トレースレコード編集機能部15は、トレースレコードファイル13内のトレースレコードと測定対象プログラム14の構成情報とを入力として編集処理を実行する。この編集処理は、まず、解析範囲指定機能

部16が編集範囲の指定を行い、次に、トレースレコード解析機能部17が解析処理を行うことにより、図2に示すような関数遷移図、図4に示すような関数呼出し関連図を作成し、さらに、トレースレコード表示機能部18が作成した図を表示するという手順により実行される。

【0017】図2に示す関数遷移図は、時系列順に実行された関数のネスト20と、関数名21と、各関数で実行されたステップ数22とを表形式で出力したものである。図2に示す例において、関数fancaは、関数fancbを呼び出すまでの間に85ステップの処理を行い、関数fancbは、関数fancaに呼び出されてから関数fanceを呼び出すまでに101ステップの処理を行い、関数fanceは、関数fancbに呼び出されてから関数fancbにリターンするまでに87ステップの処理を行っている。これ以降についての説明は省略するが、関数遷移図は、前述したように順次呼び出され、また、リターンする関数のネスト数により関連付けられて関数の遷移状態を示すと共に、その処理ステップ数を示したものである。

【0018】次に、トレースレコード解析機能部17による関数遷移図の作成動作を図3に示すフローを参照して説明する。

【0019】(1) まず、最初のトレースレコードを読み込み、トレースレコード内のアドレスと、測定対象プログラムから取得した関数名－関数アドレス対応表とを比較して、関数名を取得し、解析用テーブル30を作成する。解析用テーブル30は、解析に使用する一時情報格納用テーブルであり、図3内に示すように、関数名、表示ネスト数、実行ステップ数によるレコードにより構成され、まず、表示ネスト数＝1、実行ステップ数＝0に初期設定される(ステップ31、32)。

【0020】(2) 次のトレースレコードを読み込み、トレースレコードの取得に成功したか否かをチェックし、次のトレースレコードがない場合、この処理を終了する(ステップ33、34)。

【0021】(3) ステップ34でトレースレコードの取得に成功したと判断した場合、トレースレコード間のアドレス差分から実行ステップ数を算出し、ステップ32で作成した解析用テーブル30のレコード内の実行ステップ数に加算する(ステップ35)。

【0022】(4) ステップ33で取得したトレースレコードの分岐先アドレス、分岐命令から関数コールか否かの判定を行い、関数コールであった場合、ここまでで作成した解析用テーブルの各情報を出力する。その後、トレースレコード内の分岐先アドレスと、関数名－関数アドレス対応表とから、コール先の関数名を取得し、新たに解析用テーブル30を作成する。解析用テーブルのレコードにおける関数名以外の情報は、表示ネスト数を前の表示ネスト数＋1に、実行ステップ数を0に設定する(ステップ36、37)。

【0023】(5) ステップ36で関数コールではないと判定した場合、分岐先アドレス、分岐命令から関数リターンか否かの判定を行い、関数リターンであった場合、ここまでで作成した解析用テーブルの各情報を出力する。その後、トレースレコード内の分岐先アドレスと、関数名称ー関数アドレス対応表とから、リターン先の関数名を取得し、新たに解析用テーブル30を作成する。解析用テーブルのレコードにおける関数名以外の情報は、表示ネスト数を前の表示ネスト数ー1に、実行ステップ数を0に設定する(ステップ38、39)。

【0024】(6) ステップ39の処理終了後、及び、ステップ38で関数リターンではないと判定した場合、ステップ33からの処理を繰り返す。

【0025】関数呼出し関連図は、図4に示すように、関数遷移図を元にして、関数名称40、各関数の呼出し回数41、各関数の実行ステップ数43、自関数以下で呼出している関数の実行ステップ数の合計42を表形式で出力して、関数の呼出し関連を示した図である。また、図4においては、各関数のネスト数に応じて、その関数の実行ステップ数43、自関数以下で呼出している関数の実行ステップ数の合計42が右方にシフトされて示される。

【0026】関数呼出し関連図の作成は、図5に示すフローに従って行われる。以下、図5を参照してその処理を説明する。

【0027】(1) 最初のトレースレコードを読み込み、トレースレコード内のアドレスと、測定対象プログラムから取得した関数名称ー関数アドレス対応表とを比較して、関数名を取得し、起点ノードを作成する。作成された起点ノードを処理の対象となるノードであるカレントノードとする(ステップ51、52)。

【0028】なお、ここで使用するノードは、図5内にノード50として示すように、出現した各関数の情報(関数名、実行ステップ数等)を保持することを目的とした情報格納用テーブルであり、関数1つに対して1つ作られ、子ノードを複数持つことができる双方向リスト形式に構成されている。そして、ノード50は、前ノードに対するポインタ*pre、関数名name、呼び出し回数call、実行ステップ数(図4における43)step、総実行ステップ数(図4における42)astep、内部処理に使用する1回分の総実行ステップ数wastep、次ノードに対するポインタ*next「n」の各情報を持つ。

【0029】起点ノードの関数名以外の情報は、

- ・親(前)ノードのポインタ : 無し
- ・関数呼出し回数 : 1
- ・実行ステップ数 : 0
- ・総実行ステップ数 : 0
- ・総実行ステップ数(呼出し1回分) : 0
- ・子(次)ノードのポインタ : 全て接続無しとして設定される。

【0030】(2) 新しいトレースレコードを読み込み、トレースレコードの取得に成功したか否かをチェックし、トレースレコードが存在しない場合、解析を終了して後述する出力処理を行って処理を終了する(ステップ53、54、5D)。

【0031】(3) ステップ54で、トレースレコードが存在した場合、取得したトレースレコード相互間のアドレス差分から実行ステップ数を算出し、その値をカレントノードの実行ステップ数と、総実行ステップ数(呼出し1回分)とに加算する(ステップ55)。

【0032】(4) 分岐先アドレスや、分岐命令から関数コールか否かの判定を行い、関数コールであった場合、トレースレコード内の分岐先アドレスと、関数名称ー関数アドレス対応表とからコール先の関数名を取得し、コール先関数名をキーとして、カレントノードにコール先関数ノードが繋がっているかを検索する(ステップ56、57)。

【0033】(5) ステップ57の検索で子ノードが見つかったか否かの判定を行い、子ノードが見つかった場合、発見したノードをコール先ノードとして使用し、コール先ノードにおける情報を

- ・関数呼出し回数 : +1
- ・総実行ステップ数(呼出し1回分) : 0

に変更し、この情報変更後、カレントノードをコール先ノードに変更し、ステップ53からの処理を繰り返し実行する(ステップ58、5A)。

【0034】(6) ステップ58で子ノードが見つからなかった場合、新規にノードを作成する。新規ノードの関数名以外の情報は、

- ・親ノードのポインタ : カレントノード
- ・関数呼出し回数 : 1
- ・実行ステップ数 : 0
- ・総実行ステップ数 : 0
- ・総実行ステップ数(呼出し1回分) : 0
- ・子ノードのポインタ : 全て接続無し

として設定される。この情報設定後、カレントノードを新規ノードに変更し、ステップ53からの処理を繰り返し実行する(ステップ59)。

【0035】(7) ステップ56の判定で関数コールでなかった場合、分岐先アドレスや、分岐命令から関数リターンか否かの判定を行い、関数リターンでなかった場合、ステップ53からの処理を繰り返し実行する(ステップ5B)。

【0036】(8) ステップ5Bの判定で関数リターンであった場合、親ノードをリターン先ノードとして使用する。そして、カレントノードの変更のある情報のみの変更を行う。ここでは、カレントノードの総実行ステップ数に、総実行ステップ数(呼び出し1回分)を加算する。また、リターン先ノードの変更のある情報のみの変更を行う。ここでは、リターン先ノードの総実行ステッ

ブ数（呼出し1回分）にカレントノードの総実行ステップ数（呼出し1回分）を加算する。情報変更後、カレントノードをリターン先ノードに変更し、ステップ53からの処理を繰り返し実行する（ステップ5C）。

【0037】前述した処理を続けて、ステップ54でトレースレコードがなくなった場合、ステップ5Dによる関数呼出し関連図の出力処理が行われる。これにより、図4に示すような関数呼出し関連図が出力される。以下、図6を参照して、ステップ5Dによる出力処理の詳細を説明する。

【0038】（1）カレントノードを起点ノードに設定し、カレントノード（起点ノード）の情報を出力する（ステップ61、62）。

【0039】（2）カレントノードにデータの出力を行っていない子ノードが存在するか否かを判定する（ステップ63）。

【0040】（3）ステップ63の判定で、出力を行っていない子ノードが存在した場合、カレントノードを子ノードに移動させ、移動後のカレントノードの情報を出力し、ステップ63からの処理を続ける（ステップ64）。

【0041】（4）ステップ63の判定で、出力を行っていない子ノードが存在しない場合、親ノードが存在するか否かを判定し、親ノードが存在しない場合、出力処理を終了する（ステップ65）。

【0042】（5）ステップ65の判定で、親ノードが存在している場合、カレントノードを親ノードに移動させ、ステップ63からの処理を続ける（ステップ66）。

【0043】なお、出力処理は、前述した出力方法以外に、以下のような出力規則を持って実行される。同じノードに複数の子ノードが接続されている場合、子ノード間の出力順番は総実行ステップ数が多い順とする。関数の呼出し間連が、表の見た目で見える様に、ステップ数の表示を行う際に図4に示すように段を付けるように出力する。

【0044】次に、解析範囲の指定について説明する。前述までに説明した本発明の実施形態は、取得したトレースレコード全部を元に関数呼出し関連図を作成していたが、解析範囲指定は、まず、図2に示す関数遷移図を作成した後、この関数遷移図の中で、図7に示すように解析開始点71と解析終了点72とを指定することによって、指定された間の範囲のみで関数呼出し関連図等の作成を可能とする機能部である。

【0045】図8に示すフローを参照して、解析範囲の指定と、指定された範囲での関数呼出し関連図作成の処理動作について説明する。この処理フローは、図5により説明した関数呼出し関連図の作成の処理フローに二重線で囲んだ処理を追加したものである。

【0046】（1）まず、関数遷移図から指定された解

析開始点迄に実行される実行ステップ数（開始ステップ数）と解析終了迄に実行される実行ステップ数（終了ステップ数）とを取得して保持する。また、また、解析開始フラグと総実行ステップ数保存変数とを用意し初期値として“0”を代入する（ステップ81）。

【0047】（2）図5により説明したステップ51～54と同一の処理を順次実行する（ステップ51～54）。

【0048】（3）開始フラグが“1”とされていて、解析が開始されているか否かを判定し、開始フラグが“0”とされていて、解析が開始されていない場合、トレースレコード間のアドレス差分から、実行ステップ数を取得して総実行ステップ数に加算する（ステップ82、83）。

【0049】（4）総実行ステップ数が開始ステップ数に達したか否かを判定し、達していない場合、ステップ52の処理に戻って処理を続け、また、総実行ステップ数が開始ステップ数に達した場合、開始フラグを“1”に設定する（ステップ84、85）。

【0050】（5）ステップ82の判定で、開始フラグが“1”とされていて、解析が開始された場合、及び、ステップ85の処理で開始フラグが“1”とされた場合、図5により説明したフローにおけるトレースレコード解析処理のステップ55～5Cの処理を実行する（ステップ82、55～5C）。

【0051】（6）前述のトレースレコード解析処理の終了後、該処理から渡された実行ステップ数を総実行ステップ数に加算し、総実行ステップ数が終了ステップ数に達したか否かの判定を行う（ステップ86、87）。

【0052】（5）ステップ87の判定で、総実行ステップ数が終了ステップ数に達していなければ、ステップ53の処理に戻って処理を続け、また、総実行ステップ数が終了ステップ数に達していた場合、図6により説明した出力処理を行う（ステップ5D）。

【0053】前述した本発明の実施形態によれば、実行ステップ数の取得結果を関数の呼出し間連が理解し易い形で出力することができるので、プログラム性能解析作業を効率良く行うことができ、また、解析範囲の指定を行うことにより、実行タイミングの違いなどによる実行ステップ数の比較を行い易くすることができる。

【0054】なお、前述した本発明の実施形態は、関数単位に実行ステップ数を採取するとして説明したが、本発明は、モジュール単位に実行ステップ数を採取するようにしてもよい。

【0055】

【発明の効果】以上説明したように本発明によれば、実行ステップ数の取得結果を関数の呼出し間連が理解し易い形で出力できるようになり、作業によるプログラムの性能解析作業が効率良く行うことができる。また、解析範囲の指定を行うことができるので、実行タイミング

の違いなどによる実行ステップ数の比較も容易に行うことができる。

【図面の簡単な説明】

【図 1】 本発明の一実施形態による実行ステップ数の編集装置の全体の構成を示すブロック図である。

【図 2】 関数遷移図の構成を説明する図である。

【図 3】 関数遷移図の作成動作を説明するフローチャートである。

【図 4】 関数呼出し関連図の構成を説明する図である。

【図 5】 関数呼出し関連図の作成動作を説明するフローチャートである。

【図 6】 関数呼出し関連図の出力動作を説明するフローチャートである。

【図 7】 解析範囲の指定方法を説明する図である。

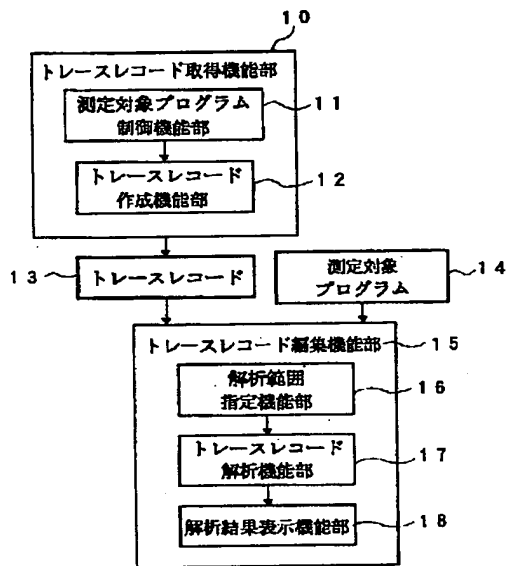
【図 8】 解析範囲を指定した解析の実行動作を説明するフローチャートである。

【符号の説明】

- 10 トレースレコード取得機能部
- 11 測定対象プログラム制御機能部
- 12 トレースレコード作成機能部
- 13 トレースレコードファイル
- 14 測定対象プログラム
- 15 トレースレコード編集機能部
- 16 解析範囲指定機能部
- 17 トレースレコード解析機能部
- 18 解析結果表示機能部

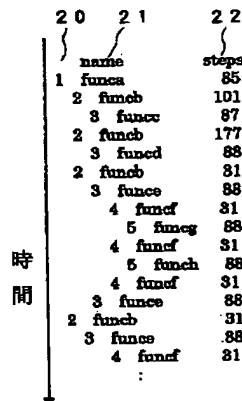
【図 1】

図 1



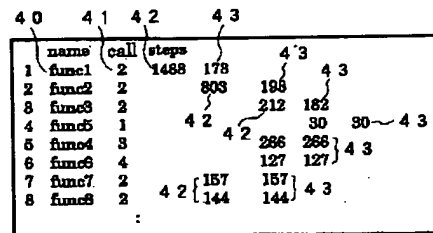
【図 2】

図 2



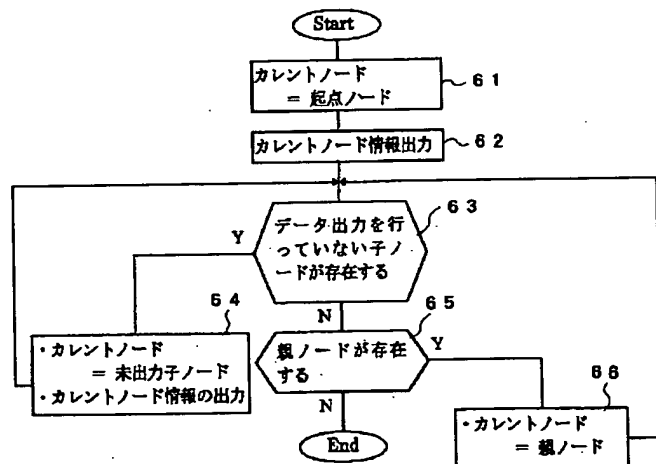
【図 4】

図 4



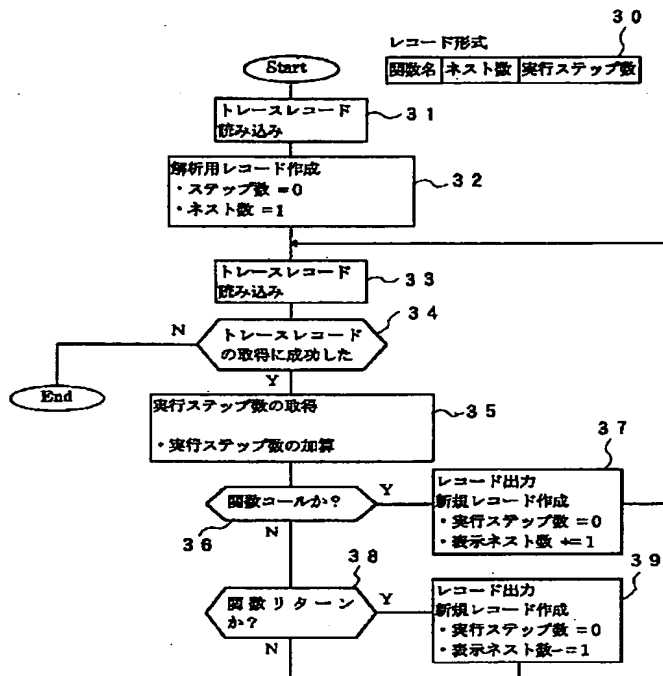
【図 6】

図 6



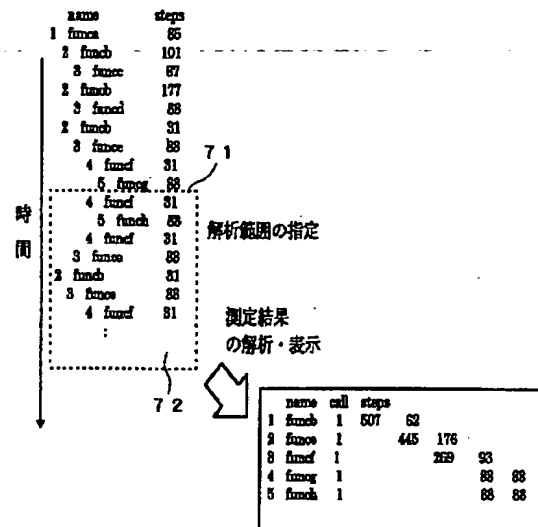
【図 3】

図 3



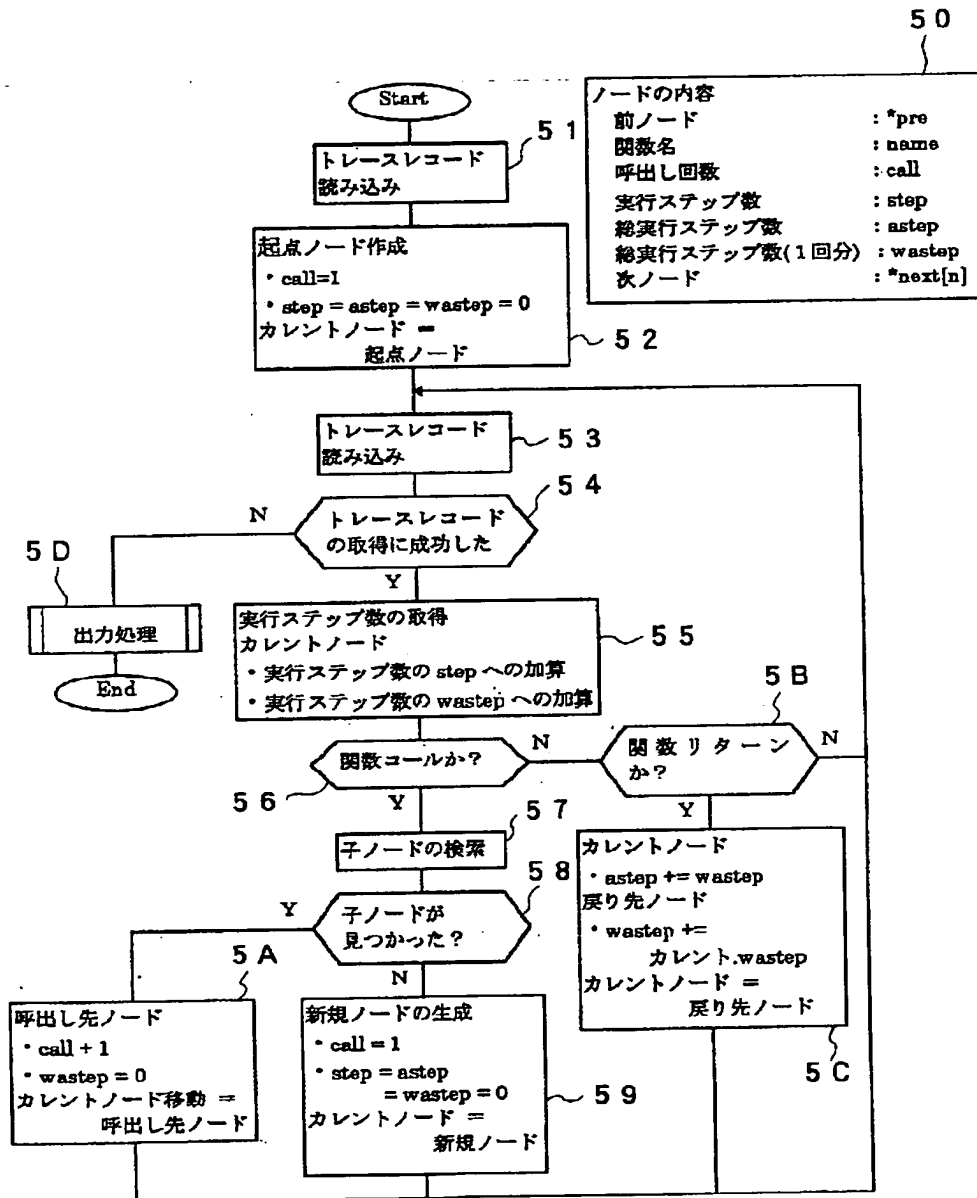
【図 7】

図 7



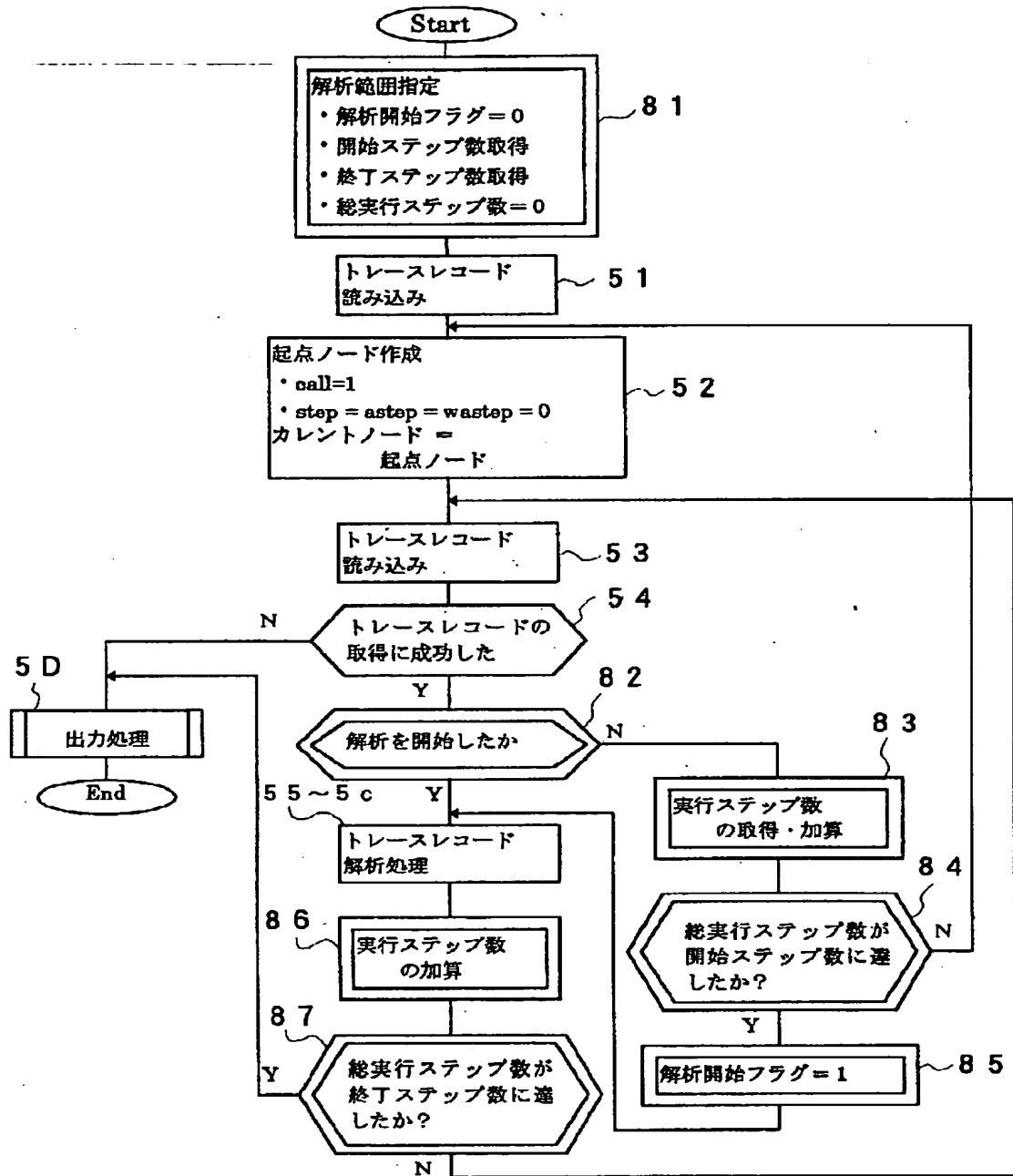
【図 5】

図 5



【図8】

図8



THIS PAGE BLANK (USPTO)